HW3 Documentation

Itamar Barron – 208981159
Roy Frumkis – 312472129

Introduction

The purpose of this project is to enhance the xv6 operating system by adding new system calls that provide information about processes to the user space. In addition, a user space program, like the "ps" command in Linux, will be developed to display the list of processes.

The project involves the implementation of three new system calls in xv6:

- **getNumProc():** This system call returns the total number of active processes in the system. It includes processes in various states such as embryo, running, runnable, sleeping, or zombie.

- **getMaxPid():** This system call returns the maximum Process ID (PID) among all currently active processes in the system.

- **getProcInfo(pid, &processInfo):** This system call takes an integer PID and a pointer to a structure called processInfo as arguments. It is used to pass information between the user and kernel modes. The system call returns 0 on success, indicating that the process information was retrieved successfully. In case of an error, such as when a process with the specified PID does not exist, a negative number is returned.

The processInfo structure, which defines the information to be returned was created, and to make this structure available to userspace programs, it is defined in a file called "processInfo.h."
With all these system calls put together, we wrote a function that iterate over all active processes in the system and print their information to the screen. For this purpose, a user space program called "ps.c" has been written, one which compiles into a user program called "ps" and compiles all the desired information regarding all active processes.

Following this header, the program is displaying a list of all active processes in the system, sorted by increasing PID number and containing this information regarding each process - PID, state, PPID, size (SZ) of process memory in bytes, number of open file descriptors (NFD), and the number of times the process was context switched in (NRSWITCH).
By implementing these new system calls and developing the "ps" user space program, users are having the ability to retrieve and display essential information about the processes running on the xv6 operating system. This functionality expands the capabilities of xv6 and allows users to monitor and understand the system's process management in a similar manner to the "ps" command in Linux.

Changes

To create this functionality, we have made changes in the origin xv6 files which we will summarize and explain; for every file we will detail the changes and elaborate on each.

Makefile -

In order to create new functionalities, we have change UPROGS (by adding _ps , _open_file and _sleep) and EXTRA (by adding ps.c open_file.c and sleep.c) so when calling the new system call the OS will be familiar with it (sleep and open_file are debug programs that will be mentioned in the end).

Proc.c -

In the proc.c file we have added the nrswitch initialization and increasing, we have located it in the scheduler(void) so every time the scheduler is switching the context it will be counted and saved.
Moreover, there are some functions we have added -

Function getNumProc: This function returns the number of active processes in the system. It iterates through the process table and increments the count for each process that is not in the UNUSED state. The variable types used are struct proc *p (pointer to a process structure) and int count.

Function getMaxPid: This function returns the maximum Process ID (PID) among all currently active processes in the system. It iterates through the process table, keeping track of the maximum PID encountered. The variable types used are struct proc *p (pointer to a process structure) and int maxPid.

Function getProcInfo: This function retrieves information about a process with a given PID and stores it in a struct processInfo pointer provided as an argument. It searches the process table for a matching PID and assigns the process state, parent PID, process sizeand number of context switches to the respective fields in the processInfo structure. It also counts the number of open files descriptor with a loop over the FOPEN list in the process struct. The variable types used are struct proc *p (pointer to a process structure), struct processInfo *pi (pointer to the process information structure), int pid, and int found.

proc.h -

This is the place we have declared all function names. We have written the function themself under the proc.c and declared them under proc.h in order to keep the order and the functionality of the code hierarchy and architecture.
First we have included the new file we have added, processInfo.h in the proc.h.
Later we have created new int variable and some int functions; nrswitch which is the number of switching and is important to the getProcInfo function, and getNumProc(void), getMaxPid(void) and getProcInfo(int pid, struct processInfo* pi which are the functions we have created in proc.c .

syscall.c -

In this section we have added the externs and their SYS calls.
Extern int for the three functions we have asked to create and their SYS calls respectevely.

Syscall.h -

In the syscall header file we have added new defines of the three asked functions and their right follow's number.

Sysproc.c -

We have added three functions -
Function sys_getNumProc: This function defines a new system call that invokes the getNumProc() function from proc.c and returns the result. It does not take any arguments. The variable types used are int for the return value.
Function sys_getMaxPid: This function defines a new system call that utilizes the getMaxPid() function from proc.c and returns the result. It does not take any arguments. The variable types used are int for the return value.
Function sys_getProcInfo: This function defines a new system call that calls the getProcInfo() function from proc.c and returns the result. It takes two arguments: int pid for the process ID and struct processInfo* pi as a pointer to the processInfo structure. The argint() and argptr() functions are used to retrieve the arguments from user space.

User.h -

We have included the struct we have asked to create under the processInfo.h file and declaired on the three new int type functions we have created:
getNumProc(void), getMaxPid(void), getProcInfo(int pid, struct processInfo* pi)

Usys.S -

In this file we have declaired out new SYS calls. Here we added the SYSCALL(func) three new line and this is what makes the functions callable.

Debug programs – open_file.c and sleep.c

sleep.c - The first program creates a parent process that forks twice, creating two child processes. Each child process prints its own process ID (PID) and sleeps for a certain duration. The parent process waits for a shorter duration and then exits. This program is used to test the "ps" function in xv6, which displays information about running processes.
open_file.c - The second program creates a parent process that forks once to create a child process. The child process opens three files, checks if the opening was successful, sleeps for 10 seconds, closes the files, deletes them, and then exits. The purpose of this program is to test file handling operations (open, close, delete) in xv6, specifically for the "ps" function.

Additional Steps

For all in all, these are all the changes we have made in the origin files in order to make our code completely work. Of course, we have wrapped this whole folder as tgz file and set everything to be aligned with the submission guidelines.